Benchmark Study on Distributed XML Filtering Using Hadoop Distribution Environment

Sanjay Kulhari, Jian Wen
UC Riverside
Team

Sanjay Kulhari  
M.S. student, CS  
U C Riverside

Jian Wen  
Ph.D. student, CS  
U C Riverside
Outline

• Pub/Sub Systems
• Project Overview and Goals
• Theoretical Core: XML Filtering
• Implementation:
  • single-threaded, multi-threaded, MR
• Experimental Evaluation
• Conclusions and Future Work.
Pub/Sub Systems: Motivation

- Think about the following Google’s services

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maps</td>
<td>View maps and directions</td>
</tr>
<tr>
<td>Images</td>
<td>Search for images on the web</td>
</tr>
<tr>
<td>Web Search</td>
<td>Search billions of web pages</td>
</tr>
<tr>
<td>Alerts</td>
<td>Get email updates on the topics of your choice</td>
</tr>
<tr>
<td>Reader</td>
<td>Get all your blogs and news feeds fast</td>
</tr>
<tr>
<td>Groups</td>
<td>Create mailing lists and discussion groups</td>
</tr>
</tbody>
</table>

Input queries and get results!  
Register your interests and get updates!
Pub/Sub Systems: Properties

- Compared with traditional search services:

<table>
<thead>
<tr>
<th>Traditional Query</th>
<th>Pub/Sub</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documents are known.</td>
<td>Queries are known.</td>
</tr>
<tr>
<td>Queries may come any time.</td>
<td>Documents are feed as a stream.</td>
</tr>
<tr>
<td>Users need quick answers. (always!)</td>
<td>Real time (a log monitoring system) or non-real time (Google Group…)</td>
</tr>
</tbody>
</table>
Architecture of a Pub-Sub System
Project Overview and Goals

- To provide distributed XML filtering with high scalability on large clusters.
  - XML filtering is a resource intensive operation.
  - Number of profiles to be matched can be huge.
  - Length of the profile can be huge.
- In our project, the scalability of the YFilter is checked.
  - Three benchmark platforms: single-threaded, multi-threaded, and map/reduce.
  - Goal: Any gains from distributing the algorithm?
Theoretical Core: XML Filtering

- Documents are matched to specified XPath queries
- Required for publish-subscribe systems
- Index is created on available subscription requests (XPath profiles)
Theoretical Core: Filtering Algorithms

- There are many existing works on filtering algorithms:
  - Software: Profiles are indexed (as finite state machine, for example).
  - Hardware: Profiles are mapped into FPGA devices.
- Our choice: YFilter
  - Parallel-able.
  - Efficient.
  - Easy to implement.
Theoretical Core: YFilter (Original)

- Profiles are indexed as a NFA in advance.
- Documents then are fed into the filter.
- The matching query is processed by traversing the NFA.
Theoretical Core: YFilter (Original)

- NFA built in YFilter
Theoretical Core: YFilter (Parallel)

- YFilter is easy to be paralleled: profiles can be divided into parts and be indexed separately.
Project Implementations

- Three benchmark platforms are implemented in our project:
  - Single-threaded: Directly apply the YFilter on the profiles and document stream.
  - Multi-threaded: Parallel YFilter onto different threads.
  - Map/Reduce: Parallel YFilter onto different machines (currently in pseudo-distributed environment).
Benchmark 1: Single Thread

- The index (NFA) is built once on the whole set of profiles.
- Documents then are streamed into the YFilter for matching.
- Matching results then are returned by YFilter.
Benchmark 2: Multiple Threads

- Profiles are split into parts, and each part of the profiles are used to build a NFA separately.
- Each YFilter instance listens a port for income documents, then it outputs the results through the socket.
Benchmark 3: Map/Reduce

- Same strategy as the multi-threaded version, however all processes are handled by Hadoop.
- Profile splitting: Profiles are read line by line with line number as the key and profile as the value.
  - Map: For each profile, assign a new key using \((\text{old\_key} \% \text{split\_num})\)
  - Reduce: For all profiles with the same key, output them into a file.
  - Output: Separated profiles, each with profiles having the same \((\text{old\_key} \% \text{split\_num})\) value.
Benchmark 3: Map/Reduce

- Document matching: Split profiles are read file by file with file number as the key and profiles as the value.
  - Map: For each set of profiles, run YFilter on the document (fed as a configuration of the job), and output the old_key of the matching profile as the key and the file number as the values.
  - Reduce: Do nothing.
  - Output: All keys (line numbers) of matching profiles.
Benchmark 3: Map/Reduce
Experimental Evaluation

- **Hardware:**
  - Macbook 2.2 GHz Intel Core 2 Duo
  - 4G 667 MHz DDR2 SDRAM

- **Software:**
  - Java 1.6.0_17, 1GB heap size
  - Cloudera Hadoop Distribution (0.20.1) in a virtual machine.

- **Data:**
  - XML docs: SIGMOD Record (9 files).
  - Profiles: 25K and 50K profiles on SIGMOD Record.

<table>
<thead>
<tr>
<th>Data</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>478416</td>
<td>415043</td>
<td>312515</td>
<td>213197</td>
<td>103528</td>
<td>53019</td>
<td>42128</td>
<td>30467</td>
<td>20984</td>
</tr>
</tbody>
</table>
Experimental Evaluation

- Since all tests are now running on a single machine, any attempts on parallel may decrease the performance.
- Although the CPU is duo core, many administrative costs may decrease the performance significantly.

**Time Cost of the Three Benchmarks**

- MR: 6.5 x 10^6 ms
- Multi: 1.2 x 10^6 ms
- Single: 0.1 x 10^6 ms
Experimental Evaluation

Time Costs for Splitting

Thousands

- Single
- 2M2R: 2S
- 2M2R: 4S
- 2M2R: 8S
- 4M2R: 4S

Time(ms)
Experimental Evaluation

There are memory failures, and jobs failed too.

Map/Reduce: # of Splits on Profiles

Time

Tasks

0:00:00 0:00:43 0:01:26 0:02:10 0:02:53 0:03:36

0 1 2 3 4 5 6 7 8 9

2 split
4 split
6 split
8 split
Experimental Evaluation

Map/Reduce: # of Mappers

Time

Tasks

2M2R
4M2R
Experimental Evaluation

There are memory failures but recovered.

Map/Reduce: # of Profiles

- Time
- Tasks

25K
50K
Interesting Stuffs

- Run-out-of-memory: We encountered this problem in all the three benchmarks, however Hadoop is much robust on this:
  - Smaller profile split
  - Map phase scheduler uses the memory wisely.

- Race-condition: since the YFilter code we are using is not thread-safe, in multi-threaded version race-condition messes the results; however Hadoop works this around by its shared-nothing run-time.
  - Separate JVM are used for different mappers, instead of threads that may share something lower-level.
Conclusion and Future Work

- **Conclusion**
  - XML pub/sub systems on large cluster is feasible.
  - Single machine tests show that no performance gains can be achieved by paralleled through threads/virtual machines.
  - Hadoop provides better framework on handling parallel and fault tolerance.

- **Future Work**
  - Tests on real distributed environment.
  - More inspection on the map/reduce framework for stream processing.
References


- YFilter: [http://yfilter.cs.umass.edu/](http://yfilter.cs.umass.edu/)

- Cloudera Hadoop Distribution: [http://www.cloudera.com/hadoop](http://www.cloudera.com/hadoop)
Questions